Developing BIST for Custom-built FPGAs

# Context

Built-in self-test (BIST) for field programmable gate arrays (FPGAs) allows reconfigurable chips to use their abundant, flexible resources to test itself with limited overhead. Configurable logic blocks (CLBs) in an FPGA are configured as test pattern generators (TPGs) and output response analyzers (ORAs), in order to test other CLBs in the FPGA. Implementation of BIST structures in FPGAs have been limited to use in commercial FPGA devices, produced by companies like Xilinx and Altera.

# Problem

Commercial FPGA configuration software has been leveraged for implementing BIST structures, but cannot be used as is for custom-built FPGA structures. Because the FPGA circuit elements and architectures differ between the custom-built FPGA and the commercial chip whose software is being used, configuration bits are more than likely in differing places, making the configuration made by the software inapplicable. Additionally, even if the architectures are similar, it is very likely that the configuration mechanism for the custom-built FPGA is different than that of the commercial tool, and thus the configuration format will not match the custom FPGA fabric. Because of this problem, there exists no easy implementation of BIST structures for custom-built FPGA fabrics.

BIST implementations for FPGAs are especially important for the research on custom FPGA development here at UVa. Currently, testing for FPGA structures has no real structure. There is no way to determine whether or not the configurations of certain logic blocks are correct, aside from observing the output. In addition, developing test vectors and inputs for the FPGA structures is currently inefficient

# Proposed Solution

In this class project, we will develop a methodology for implementing BIST structures on custom-built FPGAs. This methodology will include extending open-source FPGA mapping tools to also provide configuration bitstreams for all FPGA configurations, including BIST. The project goals are as follows:

* Find suitable TPG and ORA implementations for BLEs
* Map algorithms to FPGA fabrics using VTR open-source tool
* Extend open source FPGA CAD-tools to create BIST configuration initial conditions (IC) commands for simulation
* Proof of concept simulations of BIST tests working
* Generation of configuration bitstream for hardware
* Implementation of BIST structures on custom CLB hardware

# Prior Work

The Verilog-to-Routing tool (VTR) creates virtual mappings of a user-provided benchmark circuit or algorithm (written in Verilog) configured onto an FPGA with a user-provided architecture (described in an architecture file). The mapping of the FPGA is illustrated through a set of output files, which report the placement and connectivity of logic blocks, and the distribution of used logic resources. The tool also reports estimates for different design metrics, such as critical path delay, area, and routing resource utilization. While the tool provides a wealth of information, it is abstract and high level. There is, unfortunately, no notion of hardware or configuration bits in the provided mapping. As an example, the routing file from VTR is an output that describes routing connectivity between different logic blocks of the FPGA through the global interconnect. However, this file has no information about which values to set to any configuration bits in order to realize this mapping in hardware. For this reason, VTR and other open-source tools like it fall short for configuring FPGAs.

There have been attempts to extend the VTR tool in order to support custom-FPGA configuration. One tool, called DAGGER, takes the outputs from an older version of VTR (called VPR) and creates a configuration bit stream for FPGA architectures. The problem with this tool is that it only works for one specific FPGA, the one that those researchers custom-built. That tool would need to be altered extensively to be implemented with the FPGA structures that are being built here at UVa.

As for FPGA BIST itself, there are many examples of implementing BIST structures in literature, so I believe that finding algorithms for testing structures for the purposes of this project will be straight forward.

# Approach

Before we can map FPGAs with BIST structures, I will need to create a way to map open-source mappings of tools to FPGAs. To do that, I will extend the VTR tool in order to allow it to not only create virtual mappings for FPGAs, but provide information about the underlying circuitry, such that the tool will also be able to generate either configuration bitstreams in order to classically configure an FPGA, or to provide initial condition statements to allow me to configure the FPGA structures for simulation directly at the configuration bits, without worrying about the configuration mechanism.

Next, I will need to find Verilog algorithms to use as inputs to this new VTR-extension toolflow. I will be looking for exhaustive TPG algorithms (i.e. k-bit binary counters, where k is the number of inputs to the BLE). For ORAs, a little more work will need to be done in order to determine the necessary functionality. Hopefully, it will be as simple as borrowing from other FPGA BIST methodologies in literature. One simple way to do it would be to configure another BLE w/ the same functionality, and have a third BLE configured to compare the outputs of the BLE under test and the golden BLE.

I will finally use the toolflow that I’ve developed and map TPG and ORA configurations to different multiple BLEs, and then test other BLEs using these structures. The first level of verification will be to configure a single CLB to have one of its BLEs tested by other structures internal to the CLB. After that is completed and verified, I will move on to test larger structures.

# Results

At the end of the project, I will provide a methodology for implementing BIST structures for researches here at UVa who are looking to work on FPGA research. Not only will the new BIST structures test to make sure that configurations are correct (for the purposes of functionality testing), but also streamline testing for the purposes of collecting data. This infrastructure will include

* Verilog code for TPG and ORA implementations
* Set of IC commands for specific BLE subckts
* Verification (through simulation) of working BIST structures for FPGAs

# Milestone List

By March 3rd:

* Finalize Verilog code for TPG and ORAs
  + Either find it or write Verilog code
* Finish VTR extension, allowing for con

By March 24th

* Simulated verification of BIST structures in CLBs
* Tool-flow capability: Generation of single TPG and ORA for individual BLEs